Practically Efficient Group Signature Scheme

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Information Technology – Information Security

Brandon L. Mendrick

B.S.E., Computer Science, University of Michigan

Carnegie Mellon University
Pittsburgh, PA

December, 2019

Acknowledgements

First, I would like to thank my advisor Dr. Takanori Isobe and reader Dr. Virgil Gligor. Their input, particularly in conretely communicating ideas and concepts, was invaluable.

I would also like to thank my co-advisor Dr. Kilho Shin for sharing his knowledge on group signatures and the related areas of cryptography. Without his discussions and feedback, this paper would not have been possible.

Thank you to Jacqueline Savalle for the many late night discussions to keep me sane and for the encouragement to keep pursuing my goals. A big thank you to Tyler Seruga, as well, for helping me express and collect my thoughts for the defense presentation.

Lastly, a thank you to my numerous friends and family members. Your love and support over the years has brought me great joy, comfort, and strength. This work would not have been possible without you.

This thesis was self-funded by the researcher.

Abstract

Group signature schemes enable a set of members to anonymously sign data on behalf of the entire group. In order to prevent misuse, a designated group manager has the ability to trace a given signature back to a member. Other extensions can also be realized in specifically designed schemes, such as verifier local revocation (VLR) wherein only the verifier needs information about the validity status of signing key pairs.

This paper contributes two results towards group signature schemes. First, we present a new design for a group signature scheme that is secure under the standard model, with common relaxations for anonymity. The scheme also enables VLR and allows for fully-dynamic groups; i.e. groups that allow members to both leave and join after creation. Secondly, we implement a preliminary version of the scheme to begin investigating the efficiencies gained through utilizing one-time signing keys, instead of the traditional non-interactive, zero-knowledge proof systems.

Table of Contents

List of Tables

List of Figures

1

Introduction

In 1991, Chaum and Van Heyst proposed a novel concept called group signatures [13]. Group signatures allow for a carefully crafted set of private keys to generate digital signatures, all of which are verifiable by a single public key. Each of the private keys can be given to a different group member, allowing them to digitally sign anonymously on behalf of the group. Interestingly, it also allows a group manager the privilege of being able to ”open” any signature and link it back to a specific individual, should the need arise.

These properties and concepts were phrased somewhat informally in Chaum and Van Heyst's paper, but have come to have well established names and definitions. Those properties, excluding the obvious ones applicable to all digital signature schemes, include: group unforgability, anonymity, traceability, unlinkability, and coalition resistance. For brevity, the definitions of these security properties are left to Appendix A. Nevertheless, it is crucial to highlight that they (both the properties and authors) laid the foundation for group signatures.

1.1 Background Information

To understand nearly two decades of research, we will attempt to break the approaches down into three main periods of time: the *Early Years*, the *Efficiency Climb*, and finally the *Modern Twists.*

1.1.1 The Early Years

The *Early Years* happened directly after the release of Chaum and Van Heyst's paper. This era was defined by exploration of group signatures in order to gain an understanding of the concept. For instance, Chen and Pedersen [14] explored a slightly modified scheme that changed two properties. First, they removed traceability, meaning that the signatures were unconditionally anonymous. Second, they achieved some semblance of what is now known as a dynamic group by allowing members to join after the group was created.

We should point out that Chaum and Van Heyst's paper [13] did not clearly define group signatures. They proposed a number of possible schemes to achieve the goal, but did not show a full proof for any of them. These were the problems that Chen and Pedersen were addressing in their 1994 paper, and also the subject of Camenisch's 1997 paper *Efficient and Generalized Group Signatures* [8]. Camenisch concretely showed and proved a group signature scheme that achieved all of the necessary properties. The efficiency claim was an added bonus and something to be discussed further in the *Efficiency Climb* era. Camenisch also demonstrated the ability to adapt his scheme into threshold group signatures.

Threshold group signatures schemes embody the idea of needing more than one group member to sign on behalf of the group. Threshold signature schemes are often expressed as (t, n)-group signatures where n represents the number of potential signers (group members) and t is the number of members required to create a signature for the group [4, 8, 20, 33].

To round out the *Early Years* there are two other papers to note: one by Kim *et al.* [24] and the other by Petersen [31]. Kim *et al.* explored convertible group signatures which allow a group member to publish some small amount of data in order for all of their anonymous signatures to be converted into regular digital signatures. Petersen showed that any digital signature scheme can be converted into a group signature scheme. Although Petersen's result is interesting, the general approach that was needed made the idea inefficient.

1.1.2 The Efficiency Climb

The *Efficiency Climb* came as a result of the inefficiencies in the early group signature schemes. To be precise, the early group signature schemes suffered from public key sizes that grew linearly with the group size and general computational complexity. In other words, these schemes were mathematically beautiful, but impractical for real-world applications. In this vein, there were systems proposed which fixed the length of the keys, independent of group size. For example, this property was realized by Camenisch and Stadler in their paper *Efficient Group Signature Schemes for Large Groups* [11].

Additionally, there was quite a lot of research conducted in adapting well-known and efficient signature schemes into group signature schemes. RSA was the most common choice and showed promise in addressing both computational and memory efficiency challenges. Interestingly, most of the RSA based schemes achieved the same security properties. Where they differed was in the assumptions they made in order to prove the security of the system [1, 5, 10].

1.1.3 Modern Twists

In 2003, Bellare *et al.* published a paper which detailed a method to prove the security of group signature schemes [2]. This method marks the beginning of the *Modern Twists* era as newly proposed schemes have a well-defined, method to prove the security of their solution. As a result, this era of research, which brings us to modern day, proves the security of novel group signature properties and unique combinations of existing properties using a variety of assumptions.

First, we would like to discuss revocation and fully dynamic groups. As we mentioned, the first group signature schemes only supported a fixed group size, determined at the time of creation. Slightly later on, with Chen and Pedersen [14], for instance, we saw schemes which could add new members after the initial creation is complete. However, in order to realize fully dynamic groups (i.e. groups that can add and remove members after creation), we also need a way to revoke users, which turned out to be difficult to solve.

All of the revocation schemes, even into the *Modern Twists* era, were broadcast schemes. In other words, they required information exchange between the group manager and all members in order to revoke a user from the group. Not only is this sort of communication expensive, but the actual mechanism to realize the revocations added significant overhead as well [27].

Over time, revocation mechanisms were refined and the introduction of Verifier Local Revocation (VLR) helped to speed things along. VLR is essentially a revocation mechanism that only needs to communicate with signature verifiers, not all members of a group. As far as we are aware, Boneh and Shacham were the first to propose and implement this concept [6]. Since then, a number of other schemes have implemented this idea and taken the additional step of creating fully dynamic groups [7, 21, 30].

In addition to fully dynamic schemes there has been some research into the following areas as well:

- Using bilinear pairings as the cryptographic basis [15]
- Using lattice-based cryptography as the cryptographic basis [18, 25, 26]
- Allowing linkability of signatures from the same group member [28]

1.1.4 Application Areas

There are two main application areas for group signatures: offline payment systems and vehicle communication. Offline payment systems were invented by Chaum wherein the system prevents third parties from being able to know the time, amount paid, and recipient of a transaction [12]. In essence, the idea was to create anonymous payments. Group signature appear to be a promising way to realize such a scheme as they represent an implementation of basic properties required in the system [16, 29].

Vehicle communication is another interesting and recent application area. In particular, with the introduction of self-driving cars it is important that vehicles also communicate with each other in order to better inform their decisions. This communication, referred to as vehicle-to-vehicle (V2V), opens potential attack vectors for misuse including the loss of private data. One route being explored in order to protect the driver's private data is to use both ad hoc and long term group signature schemes to anonymize the information [19, 32].

1.2 Contribution

In this thesis we contribute two main ideas towards group signature research. First, we show the construction of a group signature scheme that is secure under the standard model with common relaxations for anonymity. The scheme also enables verifier local revocation and allows for full-dynamic groups. Importantly, the scheme is built upon one-time signing key pairs which allievate the need for non-interactive, zero-knowledge (NIZK) proof of membership systems. We believe this last point to be of particular interest since NIZK proof of membership systems are known to be computationally difficult.

Secondly, we implement a preliminary version of the scheme to begin investigating the efficiencies gained through the aforementioned use of one-time signing key pairs. Further, as we will discuss in Chapter 4, there are currently very few implementations of group signature schemes. Therefore, this is also a contribution towards making group signatures more accessible to the larger security community.

Recent Related Works

Group signatures are an active area of research, as demonstrated by the recent Katsumata and Yamada paper [22]. In their paper Katsumata and Yamada explore a new group signature scheme based on lattice cryptography. However, they faced the problem that there are no known (at the time) NIZK proof systems under the lattice cryptography area. Thus, they were able to show the definition and security of a group signature scheme under the standard model without using an NIZK system [22].

Although their work is largely similar to ours there are still some key differences. First, their scheme was for a static group and thereby also had no need for VLR. Second, they utilized indexed attribute based signatures in order to alleviate the need of NIZK proof systems. Lastly, they used lattice cryptography which has only recently gained traction within the security community[1].

[1] For example, we were only able to find one library for lattice cryptography that is still being maintained, however, it is an active area of research; `https://gitlab.com/palisade/palisade-release`

2

Security Notion

2.1 Notation

We let $\mathbb{N} = \{1, 2, 3, ...\}$ represent the set of positive integers. For a set S we let $s \xleftarrow{\$} S$ be the operation where element s is chosen uniformly at random from S. Additionally, allow $\varnothing$ to represent the empty set. We write $x \leftarrow A(a, b, ...)$ to denote function A being run with inputs $a, b, ...$ and storing the output in x. Further, we write $x \leftarrow A(a, b, ... : \mathcal{O}_1, \mathcal{O}_2, ...)$ to denote function A being run with inputs $a, b, ...$ with oracles $\mathcal{O}_1, \mathcal{O}_2, ...$ and storing the output in x. Standard notation is used for negligible ($\mathbf{negl}(\cdot)$) functions with $p(\cdot)$ representing an arbitrary polynomial function. $\langle a, b\rangle$ is used to show the arbitrary concatenation of a to b.

Finally, we need a few non-standard notations. First, some algorithms encompass a series of communications between two parties. The bulk of the algorithm will be run by the group manager (referred to as `admin`) with parts being run by other entities prefaced with `member` $\rhd$, for example, to denote that `member` is running that part of the algorithm. Secondly, there will be a need to use items in a set only once over a series of operations. This is made possible by maintaining the real set and a counterpart "used" set; however, it adds nothing to the algorithms or concepts. Therefore, we abstract this notion into the set element δ where we can write $s \overset{\$}{\leftarrow} S[\delta]$ which represents uniformly at random selecting element s of S which has not been selected previously. Lastly, we let ψ denote a set of state information that can be carried between different stages of an experiment.

2.2 PKI

The proposed scheme depends on an existing Public Key Infrastructure (PKI). Since PKI implementations are so widely prevalent in modern systems, the exact definition has been left intentionally vague. Beyond basic requirements, we ask that the PKI support two functions: `CheckCRL` and `AddToCRL`. `CheckCRL` takes a certificate as input and verifies its validity against the defined certificate revocation list (CRL), returning 1 for a success (i.e. certificate is valid) and 0 otherwise. `AddToCRL` takes the PKI's signing key sk_s and a certificate as input and adds the given certificate to the revocation list.

2.3 IND-CCA

Part of our assumptions will include the existence of an IND-CCA secure encryption scheme, $\mathcal{AE} = (\mathtt{K_e}, \mathtt{Enc}, \mathtt{Dec})$. These schemes have been shown to exist under a number of assumptions and underlying mathematics. Here we recall the definition. Consider the experiment $\mathbf{Exp}^{\mathtt{ind-cca\_b}}_{\mathcal{AE},\mathcal{A}}(k)$ for encryption scheme $\mathcal{AE}$ with security parameter $k \in \mathbb{N}$ and adversary $\mathcal{A}$, as shown in Algorithm 1.

Algorithm 1 $\mathbf{Exp}^{\mathtt{ind-cca\_b}}_{\mathcal{AE},\mathcal{A}}(k)$

```
  $(pk, sk) \leftarrow \mathtt{K_e}(k)$
  $(\psi, m_0, m_1) \leftarrow \mathcal{A}(\mathtt{choose}, pk : \mathtt{Dec}(sk, \cdot))$
  $C \leftarrow \mathtt{Enc}(pk, m_b)$
  $b' \leftarrow \mathcal{A}(\mathtt{guess}, \psi, C : \mathtt{Dec}(sk, \cdot))$
  Return $b'$
```

In this setup, a fixed bit b is chosen. The challenger generates a pair of keys (pk, sk). In the first stage, `choose`, the adversary is given pk and a decryption oracle $\texttt{Dec}(sk, \cdot)$ and outputs its state information and two message m_0 and m_1. The challenger then produces C as the encryption of message m_b using key sk. In the second stage the adversary is given its state information and access to the same decryption oracle but with the caveat she is not allowed to query C to it. At the end of the stage, the adversary outputs b' which is then returned by the experiment [23, 2].

The experiments allow the definition of the adversary's advantage as:

$$\mathbf{Adv}^{\texttt{ind-cca}}_{\mathcal{AE},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\texttt{ind-cca\_1}}_{\mathcal{AE},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\texttt{ind-cca\_0}}_{\mathcal{AE},\mathcal{A}}(k) = 1]. \tag{2.1}$$

2.4 EUF-CMA

We will also need to define an EUF-CMA signature scheme, $\mathcal{DS} = (\texttt{K}_\texttt{s}, \texttt{Vf}, \texttt{Sig})$, for use in our scheme's assumptions. Recall the definition is as follows. For the experiment $\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k)$ involving adversary $\mathcal{A}$ we have the following algorithm.

Algorithm 2 $\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k)$

```
  $(pk, sk) \leftarrow \texttt{K}_\texttt{s}(k)$
  $(m, \sigma) \leftarrow \mathcal{A}(pk : \texttt{Sig}(sk, \cdot))$
  if All of the following are true then Return 1
  1. $\texttt{Vf}(pk, \sigma, m) = 1$
  2. $m$ was not queried to Sig oracle
  else Return 0
```

In the experiment shown in Algorithm 2, the challenger begins by generating a key pair (pk, sk) utilizing security parameter k. The adversary is then given pk and the signing oracle $\texttt{Sig}(sk, \cdot)$ and is tasked with creating a forged message and signature. For the forgery to be valid, it must be verified to be correct (i.e. $\texttt{Vf}(pk, \sigma, m) = 1$) and the adversary must not have submitted m to the signing oracle. We then say that a given scheme is secure against existential unforgability under chosen message attacks if the following advantage is negligible [23, 2]:

$$\mathbf{Adv}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1]. \tag{2.2}$$

2.5 Group Signatures

In their 2003 paper Bellare, Micciancio, and Warinschi created a common, formal definition of group signature schemes [2]. Additionally, they were able to prove that all previous requirements for group signature schemes could be simplified to two strong concepts: full-anonymity and full-traceability[1]. They further showed that a scheme satisfying full-anonymity and full-traceability could theoretically exist by simply assuming that trapdoor permutations exist. Their work has been used in numerous other publications and has become the *standard model* for static group signature schemes.

The group signature scheme described in Chapter 3 utilizes their standard model. However, the scheme presented in this thesis is fully-dynamic, which means that members can join and leave the group after its creation. As such, Bellare *et al.* did propose a number of modifications to enable dynamic groups [2]. Those modifications are discussed in the following sections that define group signature notations and the security properties.

2.5.1 Syntax of Group Signature Schemes

To begin, we describe the group signature scheme as

$$\mathcal{GS} = (\mathtt{GK}, \mathtt{Join}, \mathtt{Upd}, \mathtt{GSig}, \mathtt{GVf}, \mathtt{Open}, \mathtt{Revoke}). \tag{2.3}$$

These seven algorithms are the basis for the rest of the paper and are informally defined below.

[1] Refer to Appendix A for more information

- `GK` – The group signature key generation algorithm takes a security parameter $k \in \mathbb{N}$ as input and returns the tuple $(\texttt{gpk}, \texttt{gmsk}, \texttt{gisk}, U)$ where `gpk` is the group public key, `gmsk` is the group manager's secret key, `gisk` is the group's issuing secret key, and U is the set of all user identifiers that are members of the group.

- `Join` – The group join algorithm takes a security paramter $k \in \mathbb{N}$, group's issuing secret key `gisk`, and the set of current users U as input. The algorithm will add a user to the group and help them generate an initial set of group signing keys through an interactive process. For use in later oracles, such as the `Corrupt` oracle, the algorithm will also update the group's secret keys variable **gsk**. More information about **gsk** is given below.

- `Upd` – The member update function is effectively identical to `Join`, the main difference being that already existing members of the group will use this function to interactively generate a new/fresh set of signing keys for the group.

- `GSig` – The group signing algorithm that takes a secret key $\mathbf{gsk}[i][j]$ and a message m as input and returns σ, a signature over the message.

- `GVf` – The group verification algorithm takes the group public key `gpk` and a tuple of a message and signature (m, σ) as input and returns 1 if the signature verification was successful and 0 otherwise.

- `Open` – The group manager's opening algorithm takes the group manager secret key `gmsk`, the list of valid members U, the group public key `gpk`, and a tuple of a message and signature (m, σ) as input. `Open` will return a group member's identity i on success or the symbol $\perp$ on failure.

- `Revoke` – The member revocation algorithm takes the group's issuing secret key $\texttt{gisk}$, the set of current members U, and a member's identity i as input and revokes all currently valid certificates issued to the user. Additionally, the user is removed from U but not the **gsk** construct[2].

Throughout the thesis we will also need two other functions. First, `Send` is a function which moves data between two parties over a secure channel. For example, $\texttt{Send}(\texttt{admin}, x)$ would represent sending data x to `admin`. Second, `Corrupt` is an oracle, which takes a member identity i as input and returns the private key information $\mathbf{gsk}[i]$.

On the **gsk** *Variable*

The group's secret keys, **gsk**, is defined to be a set of all information related to all members within the group. For example, $\mathbf{gsk}[i][j]$ would represent the signing key, public key, and corresponding certificates for group signing key j of group member i. The purpose of **gsk** is to provide a notation for indexing user secrets and for referencing specific key usage. The variable is not meant to exist in real-world implementations[3].

2.5.2 Fully-Dynamic Groups

Fully-dynamic groups allow members to both enter and leave at any point in time. Bellare *et al.* correctly pointed out that the leave operation is a much more delicate than the join one [2]. For instance, suppose some signature σ was generated by $\mathbf{gsk}_a[i][\delta]$ (i.e. group member i during time period a). Now consider σ being verified in time period $b > a$ and suppose that i has since been kicked out of the group.

[2] The revoked member remains in the set **gsk** in order to simulate an adversary having access to signing keys and the corresponding certificates after they have been revoked from the system.

[3] Further, we will see in Chapter 3 that it is impossible for any group member or manager to construct it.

Should the signature still verify? Bellare *et al.* concisely stated this as one of two possibilities:

1. Signatures are valid as long as the person was a member when they signed.
2. Signatures are valid as long as verification takes place when the signer is a member.

Neither answer is correct as both have their drawbacks. However, for our purpose we will pick with the second option. This choice was made simply because it is easy to understand how this would be implemented in our public key infrastructure (PKI) system: when a member leaves the group, we revoke all of their certificates[4]. However, if the use case demands it, the first option is possible to realize: when a member leaves, we only revoke the certificates from the most recent time period.

To realize a fully-dynamic group, we needed to extend some of the standard model definitions. First, we must allow **gsk** to vary over time. Although this is possible in our construction, we will not make reference to time periods since it adds little value to the scheme. For instance, the group manager may generate enough keys for a given time period to satisfy all of an adversary's requests during an experiment. Alternatively, if an adversary should require keys from multiple time periods it would be trivial for them to wait until a new time period started.

Second, we also allow `gpk` to vary over time periods. In this case, we will ignore time periods due to the construction of the scheme. In Chapter 3 we will see that `gpk` and the manager are roughly equivalent to a root certificate authority's public information [2]. As such, the scheme does not require `gpk` to vary over short time periods.

[4] Doing this in such a brute force way could break unlinkability for the user, so the practice is best avoided or done in conjunction with other revocations to obfuscate which certificates belonged to a single person. Alternatively, the group manager may no longer care about unlinkability on a revoked member's signatures.

Note, however, that this functionality is important for real-world implementations where the group/root key and certificate will have a validity period.

Finally, in fully-dynamic groups we split the usual function of `GK` into two pieces `GK` and `Join`. In general this division is representative of non-static key generation. This further allows for interactive key generation processes during the `Join` operation as well. Additionally, it implies that there is a difference in the group's secret key `gmsk` and the issuing secret key[5] `gisk` [2]. We will show how these changes have been applied to the described algorithms.

On Bellare, Shi, and Zhang's Dynamic Standard Model

A few years after Bellare, Micciancio, and Warinschi [2] published their paper on a standard model for group signatures Bellare, Shi, and Zhang [3] published an updated standard model specifically for monotonically growing group signature schemes[6]. A monotonically growing group signature scheme is one where members can be added to the group after it has been created, but they cannot be removed. Redefining the standard model seems to have come from a desire to divide the role of group manager into two: one manager for opening and one manager for issuing keys to members. This division enables a more robust security model to be generated, even allowing for the compromise or partial compromise of the managers.

We have decided not to use the updated model for a combination of the following reasons:

- As mentioned, the model allows for either manager to be partially or fully compromised. Although this can be generally beneficial, any compromise of a manager

[5] The separation of keys does two things. First, it allows for a division of labor in group and issuing managers (see Section *On Bellare, Shi, and Zhang's Dynamic Standard Model*). Second, it raises the question if `gisk` should correspond to a completely separate key.

[6] The term "dynamic" is overloaded in this context, so the notion of "monotonically growing" has been proposed to prevent ambiguity.

still results in violations of the desired security properties [3]. Therefore, the benefits of dividing the manager role appear minimal.

- As a result of dividing the managers, there are additional functions and many oracles added to the system. Although these algorithms are useful when analyzing the scheme, they fundamentally realize the same security properties presented in the static standard model.
- The authors claim that a fully-dynamic group signature scheme usually is only applicable to a niche use case (as we have discussed in Section 2.5.2). Subsequently, they decided to only include monotonically growing group signature schemes because they intended the standard model to be applicable to all situations. However, this means additional algorithms and oracles would need to be added in order to satisfy member revocation, regardless of the base model used.

However, we have integrated some of the ideas into the following sections.

2.5.3 Full Anonymity

Loosely speaking, full-anonymity is obtained when an adversary, in possesion of all group member secret keys, cannot determine the identity of some signature. The adversary is also given access to the `GSig`, `Open`, `Join`, and `Revoke` oracles in order to account for a group of any size, and the adversary being allowed to see other signatures and openings [2]. Although this level of security is desirable it is rather difficult to achieve. Additionally, it may be too strict for reasonable application areas of group signature schemes [9]. As a result, a slightly weaker notion of anonymity was developed to fit more general use cases and schemes. The notion is called *selfless-anonymity* and is the subject of the following section.

2.5.4 Selfless Anonymity

Camenisch and Groth were the first to define selfless-anonymity [9]. As mentioned previously, they used it as a relaxation of full-anonymity as they presented arguments that full-anonymity was too strict for practical applications. In particular, they argued that full-anonymity is a good definition of security for certain threat models. However, there are other, arguably more common, threat models in which full-anonymity would be overbearing and potentially prevent or limit other useful features, such as signature claiming[7]. Camenisch and Groth further argue that the weaker definition also still satisfies the informal notations of anonymity and unlinkability required by Bellare, Micciancio, and Warinschi [9, 2]

Since creating a fully-anonymous scheme is quite difficult, and because of its real-world irrelevance, the scheme defined in Chapter 3 will only satisfy selfless-anonymity. Informally, in selfless-anonymity the adversary in possession of all group member signing keys, except for the two which the experiment is run upon, cannot determine the identity of a signature. As with full-anonymity, the adversary is given access to the $\mathtt{GSig}, \mathtt{Open}, \mathtt{Join},$ and $\mathtt{Revoke}$ oracles during the experiment. We then formally define the selfless-anonymity experiment in Algorithm 3.

In Algorithm 3 we consider the experiment $\mathbf{Exp}^{\mathtt{anon-b}}_{\mathcal{GS},\mathcal{A}}(k)$ for static $b \in \{0,1\}$. The challenger generates an empty group signature scheme defined by $(\mathtt{gpk}, \mathtt{gmsk}, \mathtt{gisk}, U)$. The adversary then enters the $\mathtt{choose}$ stage and is given $\mathtt{gpk}, \mathtt{gisk},$ and U as input as well as access to the $\mathtt{GSig}, \mathtt{Open}, \mathtt{Corrupt}, \mathtt{Join},$ and $\mathtt{Revoke}$ oracles. The output of the $\mathtt{choose}$ stage is the adversary's state information ψ, two identities i_0 and i_1, and a message m. The challenger then signs message m using the secret key of member i_b. In the second stage, $\mathtt{guess}$, the adversary has access to the same oracles and is given her state information ψ and the signature σ. The adversary returns a

[7] Signature claiming is where a group member can publically claim signature they have produced.

Algorithm 3 $\mathbf{Exp}^{\mathtt{anon-b}}_{\mathcal{GS},\mathcal{A}}(k)$

```
$(\mathtt{gpk}, \mathtt{gmsk}, \mathtt{gisk}, U) \leftarrow \mathtt{GK}(k)$
$(\psi, i_0, i_1, m) \leftarrow \mathcal{A}(\mathtt{choose}, \mathtt{gpk}, \mathtt{gisk}, U : \mathtt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot),$
    $\mathtt{Open}(\mathtt{gmsk}, U, \mathtt{gpk}, (\cdot,\cdot)), \mathtt{Corrupt}(\cdot), \mathtt{Join}(k, \mathtt{gisk}, U), \mathtt{Revoke}(\mathtt{gisk}, U, \cdot))$
$\sigma \leftarrow \mathtt{GSig}(\mathbf{gsk}[i_b][\delta], m)$
$b' \leftarrow \mathcal{A}(\mathtt{guess}, \psi, \sigma : \mathtt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot), \mathtt{Open}(\mathtt{gmsk}, U, \mathtt{gpk}, (\cdot,\cdot)),$
    $\mathtt{Corrupt}(\cdot), \mathtt{Join}(k, \mathtt{gisk}, U), \mathtt{Revoke}(\mathtt{gisk}, U, \cdot))$
if All of the following are true then Return $b'$
    1. $m, \sigma$ was not queried to Open oracle during guess stage
    2. Neither $i_0$ nor $i_1$ were not queried to Corrupt during either stage
    3. Neither $i_0$ nor $i_1$ were queried to Revoke during either stage
else Return $\perp$
```

bit b' at the end of the the `guess` stage. We say that the adversary has won the experiment if $b' = b$, (m, σ) was not queried to `Open` during the `guess` stage, and neither i_0 or i_1 was queried to the `Corrupt` nor `Revoke` oracle during either stage. Note that the first win condition is checked externally, as the experiment outputs b'. However, the experiment will return $\perp$ if one of the oracle rules is violated[8].

With this scheme in mind, we say that a group signature scheme is secure under selfless-anonymity if the following advantage is negligible:

$$\mathbf{Adv}^{\mathtt{anon}}_{\mathcal{GS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathtt{anon-1}}_{\mathcal{GS},\mathcal{A}} = 1] - \Pr[\mathbf{Exp}^{\mathtt{anon-0}}_{\mathcal{GS},\mathcal{A}} = 1]. \tag{2.4}$$

Ishida *et al.* [21] aptly points out that selfless-anonymity is usually associated with two problems. First, in typical schemes under the standard model it implies that the compromise of a user's private key compromises all past and future signatures. Second, selfless-anonymity can imply that the group manager, knowing the private keys of all members, could generate signatures on behalf of the users. Although these problems are not formally defined, we will show that these concerns do not affect the scheme described in Chapter 3. Since these are not formally defined issues we coin the

[8] We assume the case of outputting $\perp$ to be trivially non-existent. An attacker is aware of the rules of the game and should not break them as that action would result in an automatic loss/failure.

term *strong selfless-anonymity* to represent a system that can demonstrate standard selfless-anonymity along with showing the above two concerns are not applicable.

2.5.5 Full Traceability

Full-traceability, as specified by Bellare *et al.* [2], is reproduced here and updated with the proper modifications to make it suitable for a dynamic group. The property ensures that only user i may generate valid signatures, all of which can be traced to user i. Informally full-traceability is defined by an adversary given the group's secret key $\texttt{gmsk}$, and working with any number of colluding group members (including the possibility of the entire group) she must not be able to create a valid signature that either cannot be opened or is not from a member of the colluding group. The adversary is also given the $\texttt{GSig}, \texttt{Corrupt}, \texttt{Join},$ and $\texttt{Revoke}$ oracles in order to simulate viewing previous signatures, corrupting group members, creating fake group members, and revoking group members, respectively. A formal definition of this property is covered by the experiment described in Algorithm 4.

Algorithm 4 $\mathbf{Exp}^{\texttt{trace}}_{\mathcal{GS},\mathcal{A}}(k)$

```
$(\texttt{gpk}, \texttt{gmsk}, \texttt{gisk}, U) \leftarrow \texttt{GK}(k)$
$(m, \sigma) \leftarrow \mathcal{A}(\texttt{gpk}, \texttt{gmsk} : \texttt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot), \texttt{Corrupt}(\cdot),$
      $\texttt{Join}(k, \texttt{gisk}, U), \texttt{Revoke}(\texttt{gisk}, U, \cdot))$
if $\texttt{GVf}(\texttt{gpk}, (m, \sigma)) = 0$ then Return 0
if $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma)) = \perp$ then Return 1
if $\exists i \in U$ s.t. the following are all true then Return 1
    1. $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma)) = i$
    2. $i$ was not queried to the oracle Corrupt
    3. $i, m$ was not queried to oracle GSig
else Return 0
```

In the experiment, the challenger generates an empty group signature scheme $(\texttt{gpk}, \texttt{gmsk}, \texttt{gisk}, U)$ and gives the adversary $\texttt{gpk}$ and $\texttt{gmsk}$ as well as access to the $\texttt{GSig}, \texttt{Corrupt}, \texttt{Join},$ and $\texttt{Revoke}$ oracles. The objective of the adversary is to output a tuple of a message and corresponding signature (m, σ). If the signature fails the verify the experiment outputs 0 signaling a failure. If the `Open` function returns $\perp$ the experiment outputs a 1 signaling success. Finally, if the `Open` function returns i such that $i \in U$, i was not queried to the `Corrupt` oracle, and i, m was not queried to the `GSig` oracle then the experiment outputs a 1. Otherwise, it returns 0.

We say a scheme is secure under full-traceability if the following advantage is shown to be negligible:

$$\mathbf{Adv}^{\texttt{trace}}_{\mathcal{GS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\texttt{trace}}_{\mathcal{GS},\mathcal{A}}(k) = 1]. \tag{2.5}$$

2.5.6 Verifier Local Revocation

Verifier Local Revocation (VLR) was originally described by Boneh and Shacham in [6]. VLR represents the ability to revoke a member by only sending revocation notifications to the verifying clients. The need for this property arose from previous schemes that required revocations to either re-key the entire group or to send a broadcast message to all members and verifiers about the event [6].

The scheme presented in Chapter 3 has this property as it is similar to current public key infrastructure (PKI) solutions. Specifically, PKIs implement some form of a certificate revocation list (CRL). These lists are maintained by the PKI "manager" and are only required by the verifiers during the verification process.

3

Constructing the Group Signature Scheme

3.1 Assumptions

The assumptions for constructing this scheme are minimal and standard. We assume the existence of an IND-CCA secure encryption algorithm $\mathcal{AE}$. Additionally, we assume that an EUF-CMA signature algorithm $\mathcal{DS}$ exists. Due to some interactive algorithms within the scheme, we require that a secure communication channel exists between the group members and the manager. This requirement is easy to satisfy in a modern setting where mechanisms such as mutually authenticated TLS can be utilized. We assume that the group manager(s) are honest[1]. Finally, the existence of a basic PKI is available for use as discussed in Section 2.2.

3.2 Definition

In this section we will define the algorithms used in our scheme. However, a few key values should be highlighted. We use $k \in \mathbb{N}$ as a security parameter. Similarly, $\lambda \in \mathbb{N}$ is used as a secondary security parameter that denotes the number of valid

[1] This idea was assumed in the static standard model ([2]) and we have further commented upon it in 2.5.2.

signing keys a user is allowed in a given time period. For both $\mathcal{AE}$ and $\mathcal{DS}$ we generate (pk, sk) key pairs and distinguish them by supscriting the variables with e or s for encryption or signing keys, respectively. `gpk` contains the group's public signing key[2] pk_s as well as the group's root certificate $\texttt{cert}_g$. `gmsk` simply contains the secret encryption key sk_e. Finally, `gisk` consists of the secret signing key sk_s and the public encryption key pk_e.

The algorithm definitions for our scheme are presented in the following sections. Each section will give the formal definition followed by a description of the algorithm.

Group Key Generation

Algorithm 5 $\texttt{GK}(k)$

```
$(pk_s, sk_s) \leftarrow \mathsf{K_s}(k)$ ; $(pk_e, sk_e) \leftarrow \mathsf{K_e}(k)$
$\texttt{cert}_g \leftarrow \texttt{Sig}(sk_s, pk_s)$
$\texttt{gpk} \leftarrow (pk_s, \texttt{cert}_g)$ ; $\texttt{gmsk} \leftarrow sk_e$ ; $\texttt{gisk} \leftarrow (sk_s, pk_e)$ ; $U \leftarrow \varnothing$
Return $(\texttt{gpk}, \texttt{gmsk}, \texttt{gisk}, U)$
```

The group key generation algorithm, shown in Algorithm 5, is used to establish a new group signature system. `GK` only takes the security parameter k as input. The algorithm generates an encryption key pair (pk_e, sk_e), a signing key pair (pk_s, sk_s), and the corresponding group certificate $\texttt{cert}_g$. The output of the algorithm is then `gpk` the group public key, `gmsk` the group manager secret key, `gisk` the group issuing secret key, and U a list of active users.

Add New Member

Algorithm 6 shows the algorithm to add a new member to the group. `Join` takes the security paramter k, the group issuing secret key `gisk`, and the list of active users U as input. Next, it uniformly at random generates a member identity i and adds it to the active user list U. Then it performs a series of operations repetitively

[2] pk_e may also be included in `gpk` but is not used outside of the group managers.

Algorithm 6 $\texttt{Join}(k, \texttt{gisk}, U)$

```
  Parse $\texttt{gisk}$ as $(sk_s, pk_e)$
  $i \xleftarrow{\$} \{0,1\}^{p(k)}$ ; $U \leftarrow U \cup \{i\}$
  for $x \leftarrow 1, ..., \lambda$ do
      $\texttt{member}_i \rhd (pk_{i,x}, sk_{i,x}) \leftarrow \texttt{K}_\texttt{s}(k)$
      $\texttt{member}_i \rhd \texttt{Send}(\texttt{admin}, pk_{i,x})$
      $\mu_{i,x} \leftarrow \texttt{Enc}(pk_e, i)$
      $\texttt{cert}_{i,x} \leftarrow \texttt{Sig}(sk_s, \langle \mu_{i,x}, pk_{i,x} \rangle)$
      $\texttt{Send}(\texttt{member}_i, \langle cert_{i,x}, \mu_{i,x} \rangle)$
      $\mathbf{gsk}[i][x] \leftarrow (k, i, \mu_{i,x}, pk_{i,x}, sk_{i,x}, \texttt{cert}_{i,x}, pk_s, \texttt{cert}_g)$
```

until λ group signing keys are generated for the new users. First, the user generates a signing key pair $(pk_{i,x}, sk_{i,x})$ and sends the public key $pk_{i,x}$ to the manager. The manager then encrypts the member identity i to create $\mu_{i,x}$. Then, the signing key certificate $\texttt{cert}_{i,x}$ is creating over the public key $pk_{i,x}$ and the encrypted identity $\mu_{i,x}$. Finally, the manager sends the certificate $\texttt{cert}_{i,x}$ and the encrypted identity $\mu_{i,x}$ back to the user[3].

Update Member Keys

Algorithm 7 $\texttt{Upd}(k, \texttt{gisk}, U, i)$

```
  Parse $\texttt{gisk}$ as $(sk_s, pk_e)$
  for $x \leftarrow 1, ..., \lambda$ do
      $\texttt{member}_i \rhd (pk_{i,x}, sk_{i,x}) \leftarrow \texttt{K}_\texttt{s}(k)$
      $\texttt{member}_i \rhd \texttt{Send}(\texttt{admin}, pk_{i,x})$
      $\mu_{i,x} \leftarrow \texttt{Enc}(pk_e, i)$
      $\texttt{cert}_{i,x} \leftarrow \texttt{Sig}(sk_s, \langle \mu_{i,x}, pk_{i,x} \rangle)$
      $\texttt{Send}(\texttt{member}_i, \langle cert_{i,x}, \mu_{i,x} \rangle)$
      $\mathbf{gsk}[i][x] \leftarrow (k, i, \mu_{i,x}, pk_{i,x}, sk_{i,x}, \texttt{cert}_{i,x}, pk_e, pk_s, \texttt{cert}_g)$
```

The algorithm for updating member keys, shown in Algorithm 7, is identical to the `Join` algorithm except for the generation of a member identity i. As a result, the description is not be duplicated.

[3] Note that **gsk** is also updated with the key information, but this is not important to the algorithm as discussed in Section 2.5.1.

Sign Data

Algorithm 8 $\texttt{GSig}(\mathbf{gsk}[i][j], m)$

```
Parse $\mathbf{gsk}[i][j]$ as $(k, i, \mu_{i,j}, pk_{i,j}, sk_{i,j}, \texttt{cert}_{i,j}, pk_e, pk_s, \texttt{cert}_g)$
$s \leftarrow \texttt{Sig}(sk_{i,j}, m)$
$\sigma \leftarrow (s, pk_{i,j}, \mu_{i,j}, \texttt{cert}_{i,j})$
Return $\sigma$
```

Algorithm 8 shows the group signature algorithm used by members to sign data on behalf of the group. GSig takes one of the member's secret keys $\mathbf{gsk}[i][j]$ and a message m as input. The algorithm then uses $sk_{i,j}$ to sign message m. Once the signature is created, the algorithm outputs $\sigma = (s, pk_{i,j}, \mu_{i,j}, \texttt{cert}_{i,j})$.

Verify Signature

Algorithm 9 $\texttt{GVf}(\texttt{gpk}, (m, \sigma))$

```
Parse $\sigma$ as $(s, pk_{i,x}, \mu_{i,x}, \texttt{cert}_{i,x})$ ;   Parse gpk as $(pk_s, \texttt{cert}_g)$
$y_1 \leftarrow \texttt{Vf}(pk_s, \texttt{cert}_g, pk_s)$
$y_2 \leftarrow \texttt{CheckCRL}(\texttt{cert}_{i,x})$
$y_3 \leftarrow \texttt{Vf}(pk_s, \texttt{cert}_{i,x}, \langle \mu_{i,x}, pk_{i,x} \rangle)$
$y_4 \leftarrow \texttt{Vf}(pk_{i,x}, s, m)$
if $y_1 = 0$ OR $y_2 = 0$ OR $y_3 = 0$ OR $y_4 = 0$ then Return 0
else Return 1
```

The group signature verification algorithm, shown in Algorithm 9, is used to verify group signatures created by group members. GVf take gpk and the signature (m, σ) as input and then performs a series of signature verifications. First, the algorithm checks that the group certificate $\texttt{cert}_g$ is valid. Next, it verifies that the member certificate $\texttt{cert}_{i,x}$ is valid, both in terms of the group's CRL and the signature. Finally, the algorithm verifies the signature s over the message m. If all of the verifications were successful, the algorithm outputs a 1. Otherwise, it outputs 0.

Open Signature

Algorithm 10 $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma))$

```
Parse $\texttt{gpk}$ as $(pk_s, \texttt{cert}_g)$ ;   Parse $\texttt{gmsk}$ as $sk_e$
Parse $\sigma$ as $(s, pk_{i,x}, \mu_{i,x}, \texttt{cert}_{i,x})$
Parse $\texttt{Dec}(sk_e, \mu_{i,x})$ as $i$
if $i \notin U$ OR $\texttt{Vf}(pk_{i,x}, s, m) = 0$ OR $\texttt{Vf}(pk_s, \texttt{cert}_{i,x}, \langle \mu_{i,x}, pk_{i,x} \rangle) = 0$ then
    Return $\perp$
else Return $i$
```

Algorithm 10 describes the signature opening algorithm used by the group manager to determine which member created a given signature. Open takes $\texttt{gmsk}, U,$ and $\texttt{gpk}$ as well as the message and signature (m, σ) as inputs. The algorithm then parses the inputs and decrypts $\mu_{i,x}$ from the certificate $\texttt{cert}_{i,x}$ included in σ. Finally, the algorithm checks that $i \in U$, the member signature on the data is valid, and the user certificate is valid. If these three checks pass, the algorithm outputs i. Otherwise, it outputs $\perp$ representing a failure state.

Revoke Member

Algorithm 11 $\texttt{Revoke}(\texttt{gisk}, U, i)$

```
Parse $\texttt{gisk}$ as $(sk_s, pk_e)$
for $x \leftarrow 1, ..., \lambda$ do
    $\texttt{AddToCRL}(sk_s, \textbf{gsk}[i][x])$
$U \leftarrow U/\{i\}$
```

The member revocation algorithm, shown in Algorithm 11, is used to revoke all of the currently active certificates of a member. Revoke takes $\texttt{gisk}, U,$ and i as inputs and retrieves the certificates pertaining to i from the certificate store. Then it uses the PKI function AddToCRL in conjunction with sk_s (from gisk) to revoke all of the retrieved certificates.

3.3 Security Analysis

Using Section 2.5 we prove the following two lemmas in order to show the security of group signature scheme $\mathcal{GS}$.

Lemma 1. *If $\mathcal{AE}$ is an IND-CCA secure ecryption scheme, then group signature scheme $\mathcal{GS}$ is selfless-anonymous.*

Lemma 2. *If $\mathcal{DS}$ is an EUF-CMA secure digital signature scheme, then group signature scheme $\mathcal{GS}$ is fully-traceable.*

We begin with a proof of Lemma 1. The proof is given in three sections: introduction, describing the adversary, and the conclusion.

Proof. We show that for any polynomial time attacker $\mathcal{B}$ who is attacking selfless-anonymity of $\mathcal{GS}$, we can construct polynomial time IND-CCA adversary $\mathcal{A}$ who attacks $\mathcal{AE}$, such that $\forall k \in \mathbb{N}$

$$\mathbf{Adv}^{\mathtt{anon}}_{\mathcal{GS},\mathcal{B}}(k) = \mathbf{Adv}^{\mathtt{ind-cca}}_{\mathcal{AE},\mathcal{A}}(k). \tag{3.1}$$

From our assumptions, we conclude that the advantage of $\mathcal{A}$ on the right hand side is negligible, therefore the left hand side is negligible as well. Importantly, the key pairs for $\mathcal{DS}$ and $\mathcal{AE}$ are generated independently. Further, all signing key pairs expect (pk_s, sk_s) are created independently across all group memebers, are only used once, and no public record exists which links either the public or secret keys to a specific member. Therefore the signature scheme and corresponding signatures are not relevant for $\mathcal{B}$ in this experiment.

Adversaries Against the Encryption Scheme

Adversary $\mathcal{A}$ is shown in Figure 3.1, described formally in Algorithm 12 and 13, and described informally in the remainder of this section.

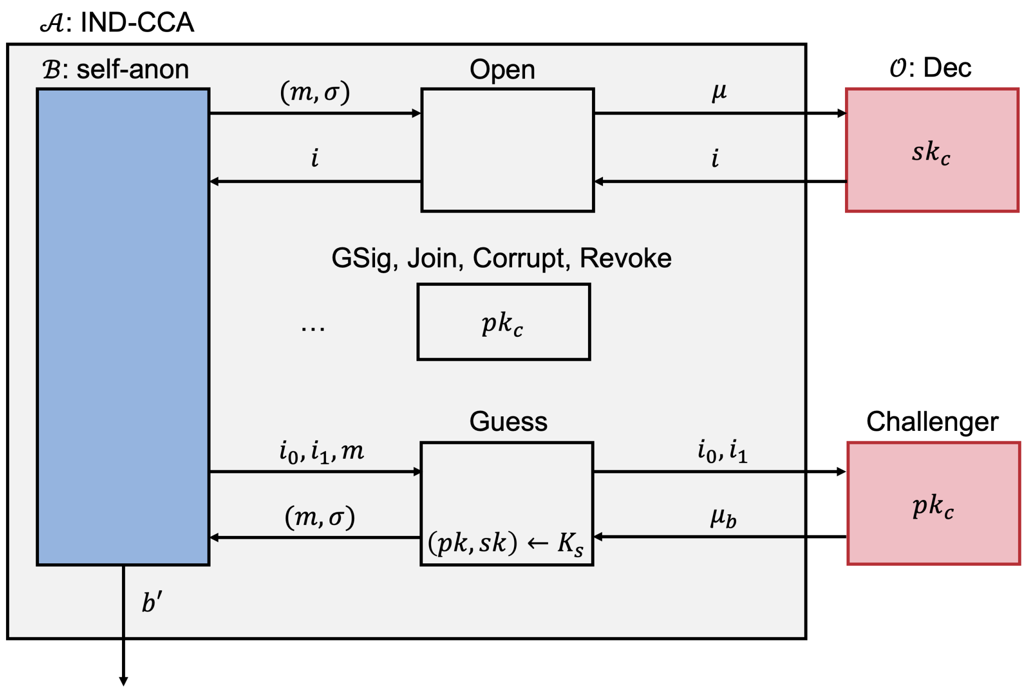

Figure 3.1: Graphic representation of the IND-CCA adversary $\mathcal{A}$ using a selfless-anonymity adversary $\mathcal{B}$.

Algorithm 12 $\mathcal{A}(\texttt{choose}, pk_c : \texttt{Dec}(sk_c, \cdot))$

```
$(pk_s, sk_s) \leftarrow \texttt{K}_\texttt{s}(k)$
$\texttt{cert}_g \leftarrow \texttt{Sig}(sk_s, pk_s)$
$\texttt{gpk} \leftarrow (pk_s, \texttt{cert}_g)$ ; $\texttt{gmsk} \leftarrow \varnothing$ ; $\texttt{gisk} \leftarrow (sk_s, pk_c)$ ; $U \leftarrow \varnothing$
$(\psi, i_0, i_1, m) \leftarrow \mathcal{B}(\texttt{choose}, \texttt{gpk}, \texttt{gisk}, U : \texttt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot),$
    $\texttt{Open}(\{\texttt{Dec}(sk_c, \cdot)\}, U, \texttt{gpk}, (\cdot,\cdot)), \texttt{Corrupt}(\cdot), \texttt{Join}(k, \texttt{gisk}, U), \texttt{Revoke}(\texttt{gisk}, U, \cdot))$
$\psi' \leftarrow (\texttt{gpk}, \texttt{gisk}, U, \mathbf{gsk}, \psi, m)$
Return $(\psi', i_0, i_1)$
```

Algorithm 13 $\mathcal{A}(\texttt{guess}, \psi', C : \texttt{Dec}(sk_c, \cdot))$

```
Parse $\psi'$ as $(\texttt{gpk}, \texttt{gisk}, U, \mathbf{gsk}, \psi, m)$ ;   Parse $\texttt{gisk}$ as $(sk_s, pk_c)$
$(pk_n, sk_n) \leftarrow Ks(k)$
$\texttt{cert}_n \leftarrow \texttt{Sig}(sk_s, \langle C, pk_n \rangle)$ ; $s \leftarrow \texttt{Sig}(sk_n, m)$
$\sigma \leftarrow (s, pk_n, C, \texttt{cert}_n)$
$b' \leftarrow \mathcal{B}(\texttt{guess}, \psi, \sigma : \texttt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot), \texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (\cdot,\cdot)),$
    $\texttt{Corrupt}(\cdot), \texttt{Join}(k, \texttt{gisk}, U), \texttt{Revoke}(\texttt{gisk}, U, \cdot))$
Return $b'$
```

For the choose stage, adversary $\mathcal{A}$ runs a modified version of $\texttt{GK}(k)$ where (pk_s, sk_s) is generated and used to create $\texttt{cert}_g$, but the keypair from the IND-CCA game is used as the encryption key. However, $\mathcal{A}$ does not have access to sk_c and must simulate access to it by use the Dec oracle. $\mathcal{A}$ then provides adversary $\mathcal{B}$ with the necessary information to start their choose stage. Note that all of the oracles for $\mathcal{B}$ can be simulated by $\mathcal{A}$, trivially. After $\mathcal{B}$ returns, the requisite information is stored in ψ' and $\mathcal{A}$ returns (ψ', i_0, i_1).

In the guess stage, adversary $\mathcal{A}$ begins by generating a new signing key pair (pk_n, sk_n) and the corresponding group certificate $\texttt{cert}_n$ for $\langle C, pk_n\rangle$. The adversary then signs $\mathcal{B}$'s requested message m with sk_n and forms the group signature $\sigma = (s, pk_n, C, \texttt{cert}_n)$. Adversary $\mathcal{A}$ now provides ψ and σ to $\mathcal{B}$'s guess stage. Once again, the oracles are easy to simulate, as above. Further, since $\mathcal{B}$ cannot query the Open oracle with m, σ, $\mathcal{A}$ need not worry about querying C to the Dec oracle. Finally, $\mathcal{A}$ returns the same answer of b' that $\mathcal{B}$ produced as her guess stage output.

Concluding the Proof

We now relate the description of adversary $\mathcal{A}$ to the formula described in the introduction section of this proof. In adversary $\mathcal{A}$'s guess stage, it establishes a perfect simulation of a group for use by $\mathcal{B}$. After receiving $\mathcal{B}$'s guess stage output, $\mathcal{A}$ outputs the same two identities to allow the IND-CCA game to act as part of the issuing manager (i.e. Join or Upd) for $\mathcal{GS}$.

When $\mathcal{A}(\texttt{guess}, ...)$ is called the first goal is to generate a signature from member i_b represented by C. The problem is, $\mathcal{A}$ is not sure which set of signing keys to use (either $\mathbf{gsk}[i_0][\delta]$ or $\mathbf{gsk}[i_1][\delta]$). As a solution, she generates a new set of signing keys (pk_n, sk_n), creates the corresponding cert $\texttt{cert}_n$ over pk_n and C, then signs m with sk_n. We argue that generating a new signing key pair is equivalent to using an existing one (from i_0 or i_1) for three reasons.

1. $\mathcal{B}$ is not given access to **gsk** so it is impossible for the adversary to know that the new key pair did not exist within the system prior to the `guess` stage.

2. Since we make no assumptions about the signature system, it is possible all key pairs are equivalent, meaning that generating a new key pair changes nothing about the scheme.

3. It would also be possible for $\mathcal{A}$ to modify **gsk** such that

$$\forall i \in U \exists x \in [\lambda] \texttt{ s.t. } \mathbf{gsk}[i][x] = (k, i, \mu_{i,n}, pk_n, sk_n, \texttt{cert}_{i,n}, pk_e, pk_s, \texttt{cert}_g) \quad (3.2)$$

   and ensure those key slots are not used during the `choose` stage of $\mathcal{B}$. This would mean that (pk_n, sk_n) is a valid key pair for all members, therefore, the key pair could be used to generate a signature over m for $\mathcal{B}$ regardless of the IND-CCA's choice of $b \in \{0, 1\}$.

Finally, $\mathcal{A}$ submits the signature of m with key pair (pk_n, sk_n) to $\mathcal{B}(\texttt{guess}, ...)$ and outputs the same result as $\mathcal{B}$. This means the probability that $\mathcal{A}$ guesses correctly is less then or equal to the probability that $\mathcal{B}$ guesses correctly in the `anon` game. Therefore, we get the equations expressed in Eq. 3.3 which satisfy the assumptions stated in 3.1.

$$\begin{aligned}
&\mathbf{Adv}^{\texttt{ind-cca}}_{\mathcal{AE},\mathcal{A}}(k) \\
&\quad = \Pr[\mathbf{Exp}^{\texttt{ind-cca\_1}}_{\mathcal{AE},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\texttt{ind-cca\_0}}_{\mathcal{AE},\mathcal{A}}(k) = 1] \\
&\quad = \Pr[\mathbf{Exp}^{\texttt{anon-1}}_{\mathcal{GS},\mathcal{B}}(k) = 1] - \Pr[\mathbf{Exp}^{\texttt{anon-0}}_{\mathcal{GS},\mathcal{B}}(k) = 1] \\
&\quad = \mathbf{Adv}^{\texttt{anon}}_{\mathcal{GS},\mathcal{B}}(k) \quad (3.3)
\end{aligned}$$

□

Next, we will prove Lemma 2. Again, the proof is given in three sections: introduction, describing the adversary, and the conclusion.

Proof. We show that for any polynomial time attacker $\mathcal{B}$ who is attacking full-traceability of $\mathcal{GS}$, we can construct polynomial time EUF-CMA adversary $\mathcal{A}$ who attacker $\mathcal{DS}$, such that $\forall k \in \mathbb{N}$

$$\mathbf{Adv}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k) \leqslant \mathbf{Adv}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k). \tag{3.4}$$

From our asusmptions, we conclude that the advantage of the adversary on the right hand side is negligible, therefore the left hand side is negligible as well. As mentioned previously, the encryption and signing key pairs are generated independently. We further argue that the use of a null encryption scheme would not affect the experiment since knowledge of member identity is unrelated to signature forgery.

Adversaries Against the Signature Scheme

Adversary $\mathcal{A}$ is shown in Figure 3.2, described formally in Algorithm 14, and informally in the remainder of this section.

Algorithm 14 $\mathcal{A}(pk_d : \mathtt{Sig}(sk_d, \cdot))$
```
    $(pk_e, sk_e) \leftarrow \mathtt{K_e}(k)$
    $\mathtt{cert}_g \leftarrow \mathtt{Sig}(sk_d, pk_d)$
    $\mathtt{gpk} \leftarrow (pk_d, \mathtt{cert}'_g)$ ; $\mathtt{gmsk} \leftarrow sk_e$ ; $\mathtt{gisk} \leftarrow (\{\mathtt{Sig}(sk_d, \cdot)\}, pk_e)$ ; $U \leftarrow \varnothing$
    $(m, \sigma) \leftarrow \mathcal{B}(\mathtt{gpk}, \mathtt{gmsk} : \mathtt{GSig}(\mathbf{gsk}[\cdot][\delta], \cdot), \mathtt{Corrupt}(\cdot), \mathtt{Join}(k, \mathtt{gisk}, U))$
    Parse $\sigma$ as $(s, pk_x, \mu_{i,x}, \mathtt{cert}_x)$
    Return $(\langle \mu_{i,x}, pk_x \rangle, \mathtt{cert}_x)$
```

Adversary $\mathcal{A}$ begins by running a modified version of $\mathtt{GK}(k)$ where an encryption key is generated and the key pair from the EUF-CMA game is used as the signing key pair for the group. Without knowledge of sk_d the adversary must utilize the $\mathtt{Sig}$ oracle to simulate its use.

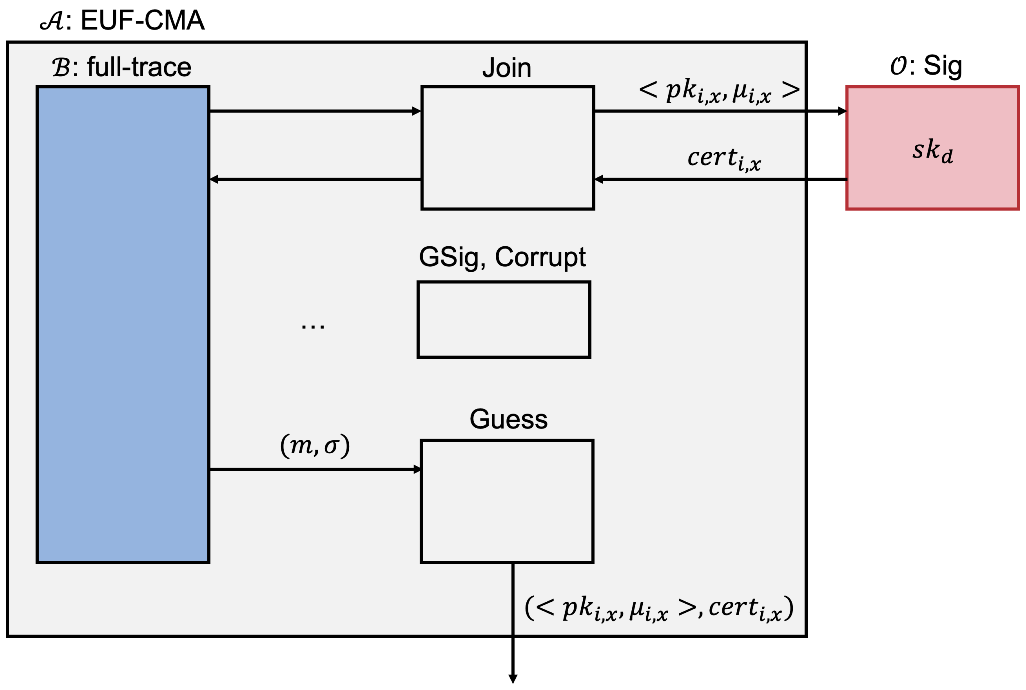

Figure 3.2: Graphic representation of the EUF-CMA adversary $\mathcal{A}$ using a full-traceability adversary $\mathcal{B}$.

Once the group signature scheme is established, $\mathcal{A}$ can call $\mathcal{B}$ with $\texttt{gpk}$, $\texttt{gmsk}$, and U. All of the oracles required by $\mathcal{B}$ are trivial for $\mathcal{A}$ to simulate. When $\mathcal{B}$ returns, $\mathcal{A}$ only needs to parse σ in order to extract the certificate signature and $(pk_x, \mu_{i,x})$ tuple over which the certificate was generated. This combination of $(\langle \mu_{i,x}, pk_x \rangle, \texttt{cert}_x)$ is then output by $\mathcal{A}$.

Concluding the Proof

We now relate the description of adversary $\mathcal{A}$ to the formula described in the first section. It should be clear that $\mathcal{A}$ is able to perfectly simulate a group signature scheme, and thereby the $\texttt{trace}$ experiment, to $\mathcal{B}$. Once $\mathcal{B}$ has given its output of (m, σ) there are three possibilities for success.

1. $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma)) = \perp$: In this case, $\mathcal{B}$ has generated a signature which we know is valid because it passed the $\texttt{GVf}$ step prior to being open. This also

tells us that the opening failed because the contained $i \notin U$. For this to be the case, $\mathcal{B}$ would have needed to generate a new, valid certificate for the fake user i by using the key pair (pk_d, sk_d). In other words, this certificate is a valid forgery for the EUF-CMA game.

2. $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma)) = i$: For this scenario, $\mathcal{B}$ has generated a valid signature for a legitimate identity i. One way this can be realized is by $\mathcal{B}$ creating a new signing key pair for member i and then generating a valid certificate using (pk_d, sk_d). Like in the scenario 1, this means that this certificate is a valid forgery for the EUF-CMA game.

3. $\texttt{Open}(\texttt{gmsk}, U, \texttt{gpk}, (m, \sigma)) = i$: This is the alternative outcome for scenario 2. Another option is for $\mathcal{B}$ to have either guessed a key of member i or have reversed the secret key from a previously seen signature by member i. These situations are nearly identical and are both a direct attack on the EUF-CMA security of the signing system.

These cases are then translated as follows:

$$\begin{aligned}\mathbf{Adv}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) &= \Pr[\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1] \\ &\geqslant \Pr[\texttt{case }1]\Pr[\mathbf{Exp}^{\texttt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1] + \Pr[\texttt{case }2]\Pr[\mathbf{Exp}^{\texttt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1] \\ &\quad + \Pr[\texttt{case }3]\Pr[\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{B}}(k) = 1] \\ &\geqslant (\Pr[\texttt{case }1]+\Pr[\texttt{case }2])\Pr[\mathbf{Exp}^{\texttt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1]+\Pr[\texttt{case }3]\Pr[\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{B}}(k) = 1].\end{aligned} \tag{3.5}$$

Since we are not aware of the probability of each case, we must evaluate the extremes and middle cases. Our three cases are as follows:

- Assume that $\Pr[\texttt{case }3] = 1$. This implies that $\Pr[\texttt{case }1] + \Pr[\texttt{case }2] = 0$. Therefore, out equation becomes:

$$\Pr[\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1] \geqslant Pr[\mathbf{Exp}^{\texttt{euf-cma}}_{\mathcal{DS},\mathcal{B}}(k) = 1]. \tag{3.6}$$

The equation should hold in that $\mathcal{B}$ should not be able to do better than $\mathcal{A}$ at attacking EUF-CMA.

- Assume that $\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}] = 1$ implying that $\Pr[\texttt{case 3}] = 0$. Note that we do not care which of the two cases happens since they realize the same attack. In this case, we get the following equation, which is identical to the desired Equation 3.4:

$$\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}} = 1] \geqslant \Pr[\mathbf{Exp}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1]. \tag{3.7}$$

- Finally, we must assume the probabilities for each case takes a value between 0 and 1. The argument utilizes the fact that $\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}] + \Pr[\texttt{case 3}] = 1$ and goes as follows:

$$\begin{aligned}\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}} = 1]&\\ &\geqslant (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}])\Pr[\mathbf{Exp}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1]\\ &+ (1 - (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}]))\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{B}}(k) = 1]. \end{aligned}\tag{3.8}$$

By moving the secondary term to the other side of the equation and utilizing our result from the $\Pr[\texttt{case 3}] = 1$ case we get:

$$\begin{aligned}\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}} = 1]&\\ &+ (1 - (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}]))\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{B}}(k) = 1]\\ \leqslant \Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}} = 1] &+ (1 - (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}]))\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1]\\ &= (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}])\Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1]\\ &\geqslant (\Pr[\texttt{case 1}] + \Pr[\texttt{case 2}])\Pr[\mathbf{Exp}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1]. \end{aligned}\tag{3.9}$$

This results in:

$$\begin{aligned}\mathbf{Adv}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) &= \Pr[\mathbf{Exp}^{\mathtt{euf-cma}}_{\mathcal{DS},\mathcal{A}}(k) = 1] \\ &\geqslant \Pr[\mathbf{Exp}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k) = 1] = \mathbf{Adv}^{\mathtt{trace}}_{\mathcal{GS},\mathcal{B}}(k). \quad (3.10)\end{aligned}$$

□

3.3.1 Strong Selfless-Anonymity

As mentioned in Section 2.5.4, there are two typical problems with group signature schemes that are only selfless-anonymous secure. In pointing out these problems, we introduced the idea of *strong selfless-anonymity* for schemes which can demonstrate these problems are not applicable as well as proving general selfless-anonymity. Below, we discuss these two problems and informally show that the scheme described in this chapter satisfies strong selfless-anonymity.

1. *In typical schemes under the standard model, selfless-anonymity implies that the compromise of a member's private key compromises all past and future signatures as well.* For the above scheme we can define a key compromise as one of two things. Either the adversary is able to gain a single secret key (i.e. $\mathbf{gsk}[i][x]$) or they are able to compromise multiple secret keys up to the entire λ keys that are issued for the time period. The first situation means the attacker can link up to a single signature back to member i since the keys are generated independently. The second situation is decidedly worse, but the same reasoning means the attacker is only able to link up to λ past or future signatures to the member i. Further, the `Upd`, `Open`, and `Revoke` functions are useful in this context to identify false signatures and compromised members, while helping to mitigate the overall security impact to the group.

2. *Selfless-anonymity can imply that the group manager, knowing the secret keys of all members, could generate signatures on behalf of the users.* In the our scheme, it is trivially clear that all secret keys for group signatures are generated by the members themselves. This means that the group manager would have to determine the secret key of the signing key pair only through knowledge of the public key, which is known to be difficult in EUF-CMA signature schemes. Admittedly, there is a possibility of the manager creating a signing key pair on their own and falsely issuing it to a member. By doing so, the group manager would have a "valid" signing key for the user. Attacks in this vein are solid arguments for dividing the role of group manager into two entities, although this brings problems of its own, as discussed in Section 2.5.2.

4

Implementation

4.1 Motivation

We implemented our scheme for two reasons. First, our solution relies heavily on utilizing an existing PKI which brings in its own computational and memory overhead. As such, we wanted to evaulate the scheme in a real-world setting. Second, we could only find a few libraries that implement group signature schemes in some capacity. The best example we could find of an existing library is the `libgroupsig` project from Jesus Diaz *et al.* [17]. The library mainly focused on standardizing an interface for all group signature implementations.

In its current state, `libgroupsig` [17] is effectively a proof of concept that never promised or showed the signature implementations to be correct, let alone optimized. The project's main ideas are discussed in Section 5.1. The paper describing to `libgroupsig` mentions a small number of other implementations. However, they implement schemes that are over a decade old.

4.1.1 Language & Algorithm Choices

When deciding on implementation paramters, we first looked at any real-world optimizations the scheme could take advantage of. In particular, we realized that an assymetric encryption scheme, while nice for mathematical proofs, was vastly more costly to compute compared to symmetric counterparts. We ended up settling on the ChaCha20 encrypting algorithm as it is now one of the recommended standards for data encryption.

For signature schemes we were concerned about the amount of data being stored on both the members and the manager, so we wanted to minimize key and certificate sizes. This notion led us quickly to elliptic curve cryptography and then onto the well-known and used NIST P-256 curve (also referred to as prime256v1). The use of elliptic curves also allowed for the possibility of key expansion as a means to limit data transfer over the communication channel, but this never came to fruition for various reasons.

In choosing our algorithms we were also careful to choose ones that were well-known and in wide-spread use. This decision allowed us to exploit existing, well-optimized libraries. To be precise, these algorithms are implemented in the popular OpenSSL cryptography stack[1].

OpenSSL is implemented in C and gives access to a command line interface. While implementing our scheme in C code next to the OpenSSL implementation would be ideal, some already complex operation (i.e. custom X.509 extensions not included in the certificate signing request) would be made overburdeningly difficult at that level. As a result, we deemed an extremely optimized implementation out-of-scope as the amount of time invested would be significant compared to the payoff. Instead, we utilized the command line interface to realize the scheme.

[1] `https://www.openssl.org/`

Finally, we must mention that the implementation utilizes X.509 certificate since they are widely used and allow for custom extensions. In our case, we utilized a custom extension on the certificates to store the encrypted user identity μ.

4.2 Our Implementation

The implementation of the scheme was done utilizing bashs scripts on a MacOS 10.14.6 machine. The scripts utilize the libreSSL 2.6.5 variant of OpenSSL. The code base is hosted in the following repository: `https://bitbucket.org/bmendrick/thesis_pegss_implementation`

4.2.1 Functions & Usage

Of the seven functions that constitute the group signature scheme, six are represented in the code base with individual executables. These six functions are described below:

- *gen_group.sh* – Equivalent to `GK` – Generates the manager's keys, the group's root certificate, and establishes the directory structure for later use.

- *create_user.sh* – Equivalent to `Join` – Adds a new user to the group. The algorithm slightly varies from `Join` in that the new member generates λ certificates before sending them all to the manager. The manager then similarly generates certificates and all λ keys before returning them to the member.

- *sign.sh <user-id> <message>* – Equivalent to `GSig` – Causes user *<user-id>* to sign the message *<message>*. Resultant signatures are stored in the *./signatures* directory.

- *verify.sh <signature> <message>* – Equivalent to `GVf` – Verifies that the signature *<signature>* is valid for the message *<message>*. This function will utilize the group/root certificate present in the signature's directory, but will retrieve an up-to-date CRL from the manager.

- *open.sh* <*signature*> <*message*> – Equivalent to `Open` – Verifies that the signature <*signature*> is valid for the message <*message*> before decrypting the user identity token on the certificate. Unlike *verify.sh*, this function does not verify the validity of the root certificate and does not check the member certificate against the CRL.
- *revoke.sh* <*user-id*> – Equivalent to `Revoke` – Searches the manager's internal pairings of user identificiations to certificate serial numbers. All certificates matching user identity <*user-id*> are then revoked (i.e. added to the CRL).

Note that the `Upd` function is missing an implementation in the scheme. While the function is valuable in long-lived groups, the testing we were looking to perform did not require its use. Further, its performance can be implied by examining the *create_user.sh* or `Join` function's performance as they are effectively identical.

4.2.2 General Metrics

The following metrics were gathered on a MacOS 10.14.6 machine running a 2.6GHz Intel i7 processor with 16GB of RAM. As mentioned above, the scripts employed the use of the OpenSSL variant libreSSL version 2.6.5. All tests were performed with a λ value of 100.

Computational Efficiency

In order to assess the computational performace of the algorithms, the `time` utility was employed and the total time elapse (i.e. "real-time") was measured. Each logical code segment was time separately with the results being stored in `time.log` files with one existing for each member, for the manager, and for the verifier.

The average time, over 10 executions, for the `GK`, `GSig`, `GVf`, `Open`, and `Revoke` algorithms are shown in Table Table 4.1. Data for the `Join` algorithm is shown in

Table Table 4.2. Note that the `Join` function is interactive between the member and manager, so it is broken into the three computation stages: member generating keys, manager generating certificates, member verifying the certificates.

Table 4.1: Average execution times for group signature algorithms

	`GK`	`GSig`	`GVf`	`Open`	`Revoke`
Time (sec)	0.050	0.047	0.036	0.046	0.117

Table 4.2: Average execution times for `Join` stages

	Member – Keys	Manager – Certs	Member – Verify
Time (sec)	1.600	3.876	1.825

When testing the `Revoke` function, we got a slightly strange result. The first execution of the function took magnitudes longer to complete than the others. Without the first outlier execution the average would be 0.012 seconds. Upon a second set of 10 trial execution, the same patterned appeared: the first execution taking significantly longer. We suspect this to be a result of files having to be loaded into memory on the first execution, but then are not paged out of memory for subsequent runs.

Memory Overhead

Memory usage is measured in two ways. First, we examine the amount of data being stored on each member as well as on the manager. This case is shown in Table Table 4.3 and shows a break down of both private and public storage. To clarify, we classify files such as private keys as needing private storage, and things such as the CRL, certificates, etc. as public[2] storage. For the manager, the size of the group and number of revoked certificates changes the storage requirements. As a baseline, we

[2] Perhaps better thought of as ”non-private” storage

have evaluated a manager with 10 active members and no revocations ($\texttt{admin}_{\text{small}}$) and one with 50 active members and 5,000 revoked certs ($\texttt{admin}_{\text{large}}$).

Table 4.3: Memory usage for members and the manager (bytes)

	`member`	$\texttt{admin}_{\text{small}}$	$\texttt{admin}_{\text{large}}$
Private	22,700	384,190	3,840,357
Public	98,500	987,428	5,114,292
Total	121,200	1,371,618	8,954,649

The second evaulation relates to the amount of traffic transferred over the secure channel by the members and manager. While this channel is only used during the `Join` and `Upd` algorithms, it is still important to understand the requirements being placed upon the network during that time. Further, knowledge of the data quantity can allow for extrapolation, based on a set of network parameters, to reason about network interaction times on either end of the connection. The amount of data transferred for a single `Join` execution are shown in Table Table 4.4.

Table 4.4: Amount of data sent over the secure communication channel during a single `Join` operation (bytes)

	`member`	`admin`
Sent	61,100	98,628
Recieved	98,628	61,100

Results

Given that there are no good comparisons, it is difficult to say whether or not the group signature scheme is efficient. From a computational point-of-view, all of the algorithms operate quickly enough for us to begin questioning the accuracy of the `time` utility's measurements. The one exception to this is the `Join` algorithm, where compute times are magnitudes larger. This, however, should not pose an issue as `Join` and `Upd` operations should be rare, realtively speaking.

From the memory usage perspective, the scheme presented in this paper clearly utilizes more memory than many of its modern counterparts. Most, if not all, modern group signature schemes use key sizes which are static or grow logarithmically in size, meaning the memory requirements for the members are a few kilobytes at most and managers have tens of kilobytes in large groups. However, storage space is inexpensive and the requirements our scheme places on the clients is not extreme, especially regarding the private data storage.

5

Conclusions

5.1 Future Work

Through the creation of this thesis, we have uncovered a number of new directions for future research. The three main areas are discussed below:

- *Group Manager(s)* – Many schemes make the assumption that group managers will act honestly. The real-world has many bad actors, however, so this assumptions may need to be relaxed or removed before group signature schemes have wider appeal. Some work has been done towards this end, for example by Bellare *et al.* [3] and their attempt to divide the manager role into two parts and allow for partial or full compromise of the managers. While allowing partial compromise is a good first step, protecting group members from full compromise of either manager would be ideal.

- *Anonymity* – Recent group signature research shows a mix of proving full-anonymity vs proving some notion of selfless-anonymity. We believe two things should happen in this area. First, we need to examine if full-anonymity and its corresponding threat model is too strong for real-world applications. Second, we need to concretely state the idea of selfless-anonymity and understand the real-world security implications that follow. These two steps will also tie into the following item about *Application Areas & Implementations.*

- *Application Areas & Implementations* – Understanding application areas will inform future research and decisions about threat models, efficiency requirements, and additional features. We believe that a key to exploring application areas is to enable security engineers to have easy access to group signature scheme implementations for use in their domains. As we have discussed in Section 4.1, however, there are currently very few group signature scheme implementations. Therefore, we are stuck in a sort of gridlock where new group signature schemes are developed without a goal, implementations are not created since new schemes are published frequently, and application areas are not explored due to a lack of implementations and the churn of new schemes. Additionally, efforts to bridge the gap between these issues, such as *libgroupsig* [17], have not gained traction. While it is unclear what a solution may be, we do believe that this cycle needs to be interrupted for group signatures to see real-world or wide spread use.

5.2 Concluding Remarks

In this thesis, we constructed a new group signature scheme and proved it to be secure under selfless-anonymity and full-traceability. These properties are representative of the standard model for group signatures, with a common relaxation for anonymity from full-anonymity to selfless-anonymity. We also introduced the notion of strong selfless-anonymity which protects against common pitfalls of group signature schemes when relaxing from full- to selfless-anonymity.

An implementation of our group signature scheme was also constructed. The intial results indicate that our scheme is computationally efficient when signing, verifying, opening, and revoking signatures. Further, we saw that the memory overhead for members was in an acceptable range. We also observed that the group manager is roughly equivalent to a certificate authority within a PKI, meaning it has acceptable computational and memory overhead as well.

Through this work we explored the creation of a group signature scheme that compromised a small amount of security to become practically efficient in use. Although this balance may not be ideal, we believe it is important to push towards schemes which are usable in the real-world in order to develop and understand application areas. Within these application areas is the future of group signature schemes and we are hopeful for their discovery and proliferation.

Bibliography

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, “A practical and provably secure coalition-resistant group signature scheme,” in *Annual International Cryptology Conference*. Springer, 2000, pp. 255–270.

[2] M. Bellare, D. Micciancio, and B. Warinschi, “Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 614–629.

[3] M. Bellare, H. Shi, and C. Zhang, “Foundations of group signatures: The case of dynamic groups,” in *Cryptographers Track at the RSA Conference*. Springer, 2005, pp. 136–153.

[4] A. Boldyreva, “Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme,” in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.

[5] D. Boneh, X. Boyen, and H. Shacham, “Short group signatures,” in *Annual International Cryptology Conference*. Springer, 2004, pp. 41–55.

[6] D. Boneh and H. Shacham, “Group signatures with verifier-local revocation,” in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 168–177.

[7] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth, “Foundations of fully dynamic group signatures,” in *International Conference on Applied Cryptography and Network Security*. Springer, 2016, pp. 117–136.

[8] J. Camenisch, “Efficient and generalized group signatures,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1997, pp. 465–479.

[9] J. Camenisch and J. Groth, “Group signatures: Better efficiency and new theoretical aspects,” in *International Conference on Security in Communication Networks.* Springer, 2004, pp. 120–133.

[10] J. Camenisch and M. Michels, “A group signature scheme based on an RSA-variant,” *BRICS Report Series*, vol. 5, no. 27, 1998.

[11] J. Camenisch and M. Stadler, “Efficient group signature schemes for large groups,” in *Annual International Cryptology Conference.* Springer, 1997, pp. 410–424.

[12] D. Chaum, “Blind signatures for untraceable payments,” in *Advances in cryptology.* Springer, 1983, pp. 199–203.

[13] D. Chaum and E. Van Heyst, “Group signatures,” in *Workshop on the Theory and Application of of Cryptographic Techniques.* Springer, 1991, pp. 257–265.

[14] L. Chen and T. P. Pedersen, “New group signature schemes,” in *Workshop on the Theory and Application of of Cryptographic Techniques.* Springer, 1994, pp. 171–181.

[15] X. Chen, F. Zhang, and K. Kim, “A new ID-based group signature scheme from bilinear pairings.” *IACR Cryptology ePrint Archive*, vol. 2003, p. 116, 2003.

[16] X. Chen, G. Lenzini, S. Mauw, and J. Pang, “A group signature based electronic toll pricing system,” in *2012 Seventh International Conference on Availability, Reliability and Security.* IEEE, 2012, pp. 85–93.

[17] J. Diaz, D. Arroyo, and F. de Borja Rodríguez, “libgroupsig: An extensible c library for group signatures.” *IACR Cryptology ePrint Archive*, vol. 2015, p. 1146, 2015.

[18] S. D. Gordon, J. Katz, and V. Vaikuntanathan, “A group signature scheme from lattice assumptions,” in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2010, pp. 395–412.

[19] J. Guo, J. P. Baugh, and S. Wang, “A group signature based secure and privacy-preserving vehicular communication framework,” in *2007 Mobile Networking for Vehicular Environments.* IEEE, 2007, pp. 103–108.

[20] L. Harn, “Group-oriented (t, n) threshold digital signature scheme and digital multisignature,” *IEE Proceedings-Computers and Digital Techniques*, vol. 141, no. 5, pp. 307–313, 1994.

[21] A. Ishida, Y. Sakai, K. Emura, G. Hanaoka, and K. Tanaka, "Fully anonymous group signature with verifier-local revocation," in *International Conference on Security and Cryptography for Networks.* Springer, 2018, pp. 23–42.

[22] S. Katsumata and S. Yamada, "Group signatures without NIZK: From lattices in the standard model," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2019, pp. 312–344.

[23] J. Katz and Y. Lindell, *Introduction to Modern Cryptography.* CRC Press, Taylor & Francis, 2015.

[24] S. J. Kim, S. J. Park, and D. H. Won, "Convertible group signatures," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 1996, pp. 311–321.

[25] F. Laguillaumie, A. Langlois, B. Libert, and D. Stehlé, "Lattice-based group signatures with logarithmic signature size," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2013, pp. 41–61.

[26] A. Langlois, S. Ling, K. Nguyen, and H. Wang, "Lattice-based group signature scheme with verifier-local revocation," in *International Workshop on Public Key Cryptography.* Springer, 2014, pp. 345–361.

[27] B. Libert, T. Peters, and M. Yung, "Group signatures with almost-for-free revocation," in *Advances in Cryptology–CRYPTO 2012.* Springer, 2012, pp. 571–589.

[28] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *Australasian Conference on Information Security and Privacy.* Springer, 2004, pp. 325–335.

[29] G. Maitland and C. Boyd, "Fair electronic cash based on a group signature scheme," in *International Conference on Information and Communications Security.* Springer, 2001, pp. 461–465.

[30] M. N. S. Perera and T. Koshiba, "Fully dynamic group signature scheme with member registration and verifier-local revocation," in *International Conference on Mathematics and Computing.* Springer, 2018, pp. 399–415.

[31] H. Petersen, "How to convert any digital signature scheme into a group signature scheme," in *International Workshop on Security Protocols.* Springer, 1997, pp. 177–190.

[32] A. Wasef and X. Shen, “Efficient group signature scheme supporting batch verification for securing vehicular networks,” in *2010 IEEE International Conference on Communications*. IEEE, 2010, pp. 1–5.

[33] Y.-L. Yu and T.-S. Chen, “An efficient threshold group signature scheme,” *Applied Mathematics and Computation*, vol. 167, no. 1, pp. 362–371, 2005.

Appendix A

Generalized Group Signature Properties

As mentioned in this thesis, Bellare, Micciancio, and Warinschi's seminal paper [2] on group signature modeling stated that their definitions of full-anonymity and full-traceability included all previous, loosely defined security properties. In Table A we list the security properties that Bellare *et al.* discussed as well as what property they follow from and a brief description. However, for an in-depth understanding these properties and how exactly they are included in the model, we refer to reader to Section 3 of [2].

Name	**Covered By**	**Description**
Unforgability	Full-Traceability	It is computationally infeasible to produce a signature (m, σ), which verifies correctly, without knoweldge of the secret keys.
Exculpability	Full-Traceability	No member of the group, not even the group manager, can produce signatures on behalf of other users.
Traceability	Full-Traceability	Originally referred to the functional property that if a message is signed by the private key of i and the `Open` algorithm is applied to it, then i must be returned. Traceability was also overloaded with a security property which later became known as coalition resistance.
Coalition Resistance	Full-Traceability	The possibility of group members colluding together in order to create signatures which could not be traced to any of them.
Framing	Full-Traceability	A variation on coalition resistance where a group of members combine their keys to produce a valid signature which opened to another member's identity.
Anonymity	Full/Selfless-Anonymity	Similar to full- or selfless-anonymity, but the attacker does not have access to the `Open` oracle.
Unlinkability	Full/Selfless-Anonymity	After seeing a list of signatures an adversary cannot relate two signatures as being produced by the same member. More generally, unlinkability is hard to define as it is highly dependent on the threat model assumptions, however, all reasonable definitions follow from full-anonymity[1].

Table 1.1: Overview of past group signature security properties

[1] Selfless-anonymity trivially covers most, but not all, of the threat models considered. Further analysis is required to understand which threat models are not covered (see item *Anonymity* in Section 5.1).